Case Studies for Self-Organization in Computer Science

Marco Mamei[1], Ronaldo Menezes[2], Robert Tolksdorf[3] and Franco Zambonelli[1]

[1] Dipartimento di Scienze e Metodi dell'Ingegneria, Universitá di Modena e Reggio Emilia
Via Allegri 13, 42100 Reggio Emilia, Italy
{marco.mamei,franco.zambonelli}@unimore.it

[2] Florida Tech, Department of Computer Sciences
150 W. University Blvd., Melbourne, FL 32901, USA
rmenezes@cs.fit.edu

[3] Freie Universität Berlin, Institut für Informatik, AG Netzbasierte Informationssysteme,
Takustr. 9, D-14195 Berlin, Germany
research@robert-tolksdorf.de

Abstract. Self-Organization is bound to greatly affect computer science. The simplicity and yet power of self-organized models will allow researchers to propose efficient solutions to problems never before thought possible to be addressed efficiently. The published works in the field clearly demonstrate the potential of this approach. This paper first reviews a number of interesting self-organization phenomena found in nature, then it discusses their potential applicability in several computer science application scenarios.

1 Introduction

Self-organization can be defined as a process in which the internal level of organization of a *system* increases automatically without being guided or managed by an outside source [74]. Self-organization has been discussed, since long, in a number of disciplines ranging from physics to biology and recently, the popularity of the term has increased and it has even appeared in the popular vocabulary. We can currently observe a swarm — to paraphrase the term — of books being published to explain this, so called by the authors, fantastic phenomenon [4, 11, 62, 70, 33, 67]. These books are in addition to the few books in the field discussing the topic from a more academic perspective [36, 8, 37, 14, 19]. In the last few years, we have witnessed the birth of nature-inspired self-organization also as a field of computer science. The reason for this upspring is easy to state: the dynamism of the new IT scenarios (like worldwide Internet computing and pervasive and ubiquitous computing) makes it impossible to think in advance of the complex chain of interactions, and the cascading side effects arising in complex distributed applications. For this reason, the complexity of software applications can no longer be managed with traditional top-down techniques. Instead, components must

be able to take care of themselves and handle unanticipated dynamic situations. The self-organizing phenomena found in nature are a perfect example of these capabilities. Insect colonies and biological cells are able to organize their activities and achieve tasks with exceptional robustness despite harsh and dynamic environments.

The aim of this paper is to present a review on the state of the art of nature-inspired self-organizing mechanisms in computer science. Accordingly, the paper is divided as follows: in Section 2 we discuss on the fundamental role of interaction mechanisms with regard to self-organization. Moreover, we present a taxonomy, to classify such interaction mechanisms that emphasizes their role in supporting self-organization. In Section 3 we present, with the help of the introduced taxonomy, a number of nature-inspired self-organizing metaphors. In Section 4, we present some of the main research efforts trying to apply self-organization concepts to develop computer science applications. Finally, in Section 5 we present our concluding remarks.

2 The Fundamental Role of Interaction Mechanisms in Self-Organization

Generally speaking, natural self-organizing systems are constituted by a large number of individuals who interact and coordinate to achieve tasks exceeding their capabilities as single individuals. Accordingly, the way in which the individuals interact is of paramount importance and most of the benefits involved in self-organization are actually due to the right way of interacting. With this regard, we want to emphasize that when we talk about interaction and coordination — as will be clear from the examples in the next section — we refer to any kind of mechanisms (e.g. with or without an explicit communication being involved) allowing some individuals to orchestrate and influence each other's activities. In this section, we want to study the space of possible interaction mechanisms to find which are more suited to support self-organizing activities. Moreover, we want to classify the interaction mechanisms according to a specific taxonomy in order to illustrate their strengths and weaknesses. This study and the resulting taxonomy will ground the discussion in the next section where we will show how different self-organizing phenomena naturally derive and are based upon specific interaction mechanisms.

Among all the possible interaction mechanisms, we will focus only on uncoupled and anonymous ones. Uncoupled and anonymous interaction can be defined by the fact that the two interaction partners (e.g. sender and recipient of an information) need neither to know each other in advance, nor to be connected at the same time in the network. For example, an interaction based on a sort of *message in a bottle* (where an agent leaves somewhere a message that is eventually retrieved by another agent) clearly belongs to this category. Uncoupled and anonymous interaction has many advantages. It naturally suits open systems, where the number and the identities of components are not known at design time. It suits also dynamic scenarios in that components can interact

also in presence of network glitches an temporary failures. Moreover, it fosters robustness in that components are not bound to interact with prespecified partners (that could possibly broke down), but interact with whoever is available. Summarizing, uncoupled and anonymous interaction is suited in those dynamic scenarios where an unspecified number of possibly varying components need to coordinate and (self-)organize their respective activities. Not surprisingly, this kind of interaction is ubiquitous in nature; cells, insects and animals adopt it to achieve a wide range of self-organizing behaviors (see next section for concrete examples). The above cited *message in a bottle* is just an example of this kind of mechanisms and many more are conceivable (and actually exploited in natural systems!). In order to proceed systematically, it is useful to create a taxonomy where to organize the various approaches. In particular, we think that a taxonomy based on the answers to the following three questions: (i) what information is communicated, (ii) what is the flow of information, (iii) how information is used, not only provides a good coverage of the possible uncoupled and anonymous interaction mechanisms, but can highlight strengths and weaknesses of these approaches to support self-organizing systems.

What information is communicated?

- *Marker-based Interactions:* the information being communicated consists of special markers that the agents use explicitly for the purpose of interaction. This is a kind of explicit interaction mechanism: actions (e.g. emitting a sound or a pheromone) are explicitly taken in order to communicate.
- *Sematectonic Interactions*: the information being communicated is the current state of the solution[1]. This is a kind of implicit interaction: the local configuration of what is being done guides the actions. For example, when sorting some items, the fact of placing similar items at a certain spot is not only a part of the solution, it is also a "signal" saying: "this kind of stuff goes there".

Sematectonic interaction is inherently *cheaper* than marker-based one. In fact, while in marker-based mechanisms, the interaction is an additional task that the agents have to undertake (i.e., they have to actively produce some marker), in sematectonic mechanisms, the interaction is embedded (comes for free) in the actual problem-solving task. On the other hand, marker-based interaction is more expressive than sematectonic one, in that the interaction-part can arbitrarily describe the problem-solving activities. In other words, while it is always possible to recreate a sematectonic interaction adopting a marker-based approach (e.g., markers can describe the current state of the solution), the opposite is not true (e.g., with markers is it possible to differentiate two solutions' states that look the same). Marker-based approaches are rooted on a shared communication facility (e.g., shared data space) where agents can post and retrieve information. Sematectonic interaction requires, instead, that the solution space is shared and observable by the agents.

[1] We assume that the agents interact to achieve a particular goal. The state of the solution of this goal is the content being communicated.

What is the flow of information?

- *Serendipitous*: following this interaction mechanism, agents produce data and leave it in the environment (i.e., in some kind of data-space shared among the agents). Other agents need to proactively look for that information to retrieve it. For example, as it will be detailed in the following, in the Linda approach [26], agents can leave tuples, containing information, in a shared data space, and can access tuples left by other agents through a pattern-matching mechanism. In this approach there is not an explicit way to direct a message to a specific recipient. The communication happens serendipitously.
- *Diffusion*: here the data is diffused across the environment to be available by a large number of agents. This is a sort of broadcast "communication", the information is sent to whoever is listening.

It is rather clear that an interaction mechanism based on the diffusion of information requires a communication infrastructure (whether natural or artificial) more powerful than a serendipitous mechanism. In the diffusion case, in fact, data need to be transmitted to where the agents are. In the serendipitous approach, data can remain confined within a specific storage facility waiting for the agents to come by.

How information is used?

- *Trigger-based*: the reading of specific data triggers some actions on the agent. This is a kind of event notification and is the standard approach to agent coordination. For example, to coordinate movements, an agent can interact with another agent by leaving a message with something like "go to waypoint (100,100)". On the basis of the information received, the latter agent can proceed to the new target.
- *Follow-through*: the interaction is not based on individual events triggering "one-shot" reactions. Instead the information being communicated drives the agent actions step-by-step. For example, to coordinate movements, an agent can leave a trail to be followed by another agent later on. In this way, the perception of the trail does not trigger single reactions, but constantly guides the agent activities.

It is important to emphasize that all the above answers are not a complete and exhaustive list of all the possible alternatives. They are just landmark points and also intermediate and mixed approaches are conceivable.

On the basis of the three questions, we can classify and represent different interaction mechanisms in a 3-dimensional space (see Figure 1). In the following section, we will show how most of the identified self-organizing strategies in nature fit well in this taxonomy.

3 Self-Organization Metaphors

Self-organization is a phenomenon that takes place in several biological multi-agent systems. Social insects, by self-organizing their activities, achieve goals that overcome by far their capabilities as single individuals [9, 54]. Other animals also demonstrate self-organized behavior. For instance large herds are completely decentralized and not influenced by any kind of leader, and still are able to perform complex coordinated activities. Most importantly for our discussion is the fact that a lot of the self-organized phenomena found in nature provide a useful inspiration to solve computer science problems. In the following we will survey different self-organization mechanisms showing how they fit in the sketched survey-space. Moreover, we will present some remarks about the possible impact of the self-organized behaviors in computer science, although a complete discussion of this topic will be presented in section 4.

3.1 Foraging

Foraging is the metaphor in which individuals self organize their activities to stochastically explore some environment in search of food (or other resources) [9, 54]. Once food is found the individuals return to their nest leaving a pheromone path behind, in the environment. The existence of pheromones, has an influence on the stochastic decision of other forager individuals — the more pheromone available the more likely an individual crossing the pheromone trail will choose that path. The pheromone is volatile and disappears from the environment after some time. This is negative feedback and results in old paths to food being forgotten if not exploited anymore. The foraging mechanism is commonly exploited by several insect colonies such as ants — *anoplolepis gracilipes* — and termites.

Considering the introduced taxonomy, foraging is rooted on a *marker-based, serendipitous and follow-through* interaction mechanism. It is *marker-based* in that the pheromones emitted by the individuals are explicitly conceived to enable interactions. It is *serendipitous* in that an individual wanders randomly until it crosses by chance a pheromone trial. It is *follow-through* in that, sensing a pheromone, does not trigger one-shot reactions, but pheromones form distributed structures (i.e. trails) to be continuously followed by individuals.

The foraging mechanism finds natural applications in a wide number of computer science scenarios, ranging from network routing to optimization algorithms. As it will be described in the next section, serval research projects use this self-organized mechanism to address computer science problems.

3.2 Nest Building

Another interesting metaphor of self organization is the mechanism by which a swarm of individuals can cooperatively build a nest or another complex structure. The building, rather surprisingly, can be realized without the presence of any cental engineer or master plan, but relying on an effective interaction mechanism and some simple local rules followed by the individuals. In a wide range of natural systems the building process is performed accordingly to this schema:

1. individuals create small "bricks" (typically made of bodily waste), which contains pheromones.
2. They wander randomly, but prefer the direction of the strongest local pheromone concentration.
3. At each step, they decide stochastically whether to deposit the current brick.
4. The probability of making a deposit increases with the local pheromone density.

This algorithm leads to the generation of scattered initial deposits. These deposits attract individuals and increase the probability that these individuals will leave a brick there. The most recent deposits are placed at the center of the pile. So, they are the strongest and piles tend to grow vertically rather than horizontally, forming columns. When columns grow near, the scent of each attracts individuals of the other, thus pulling subsequent deposits into the shape of an arch. For example, tropical termites can build nests more than 15 feet high and 10 tons in weight. These nests are multi-story structures providing storage for food, housing the brood, and protection for the population.

Considering the introduced taxonomy, nest-building is rooted on a *marker-based, serendipitous, trigger-based* interaction mechanism. It is *marker-based* in that the pheromones left by individuals in bricks are explicitly conceived to enable interactions. It is *serendipitous* in that individuals wander randomly until they find pheromones. It is *trigger-based* in that pheromones do not form paths that constantly drive individual activities, but the presence of single pheromones triggers (probabilistically) the release of the brick.

This kind of mechanism, as detailed in the next section, finds natural applications in robotics — i.e., robots collectively building some artifact.

3.3 Molding

Molding is the metaphor where individuals are attracted by specialized food sources (or resources, in general) available in the environment [60]. If food is plentiful the individuals wander autonomously looking for food. When food becomes scarce, the individuals move toward one another forming a cluster. Then the cluster (as a single super cell) begins crawling the environment looking for a more favorable spot. When

it finds such a place, it differentiates again, individuals are re-created and a new cycle begins. During the phase of the molding (aggregation) individuals produce a specialized chemical that diffuse in the neighborhood. Individuals are also attracted by the same pheromone. They follow a gradient of the chemical, tending to aggregate where the chemical is the strongest. The molding process is exploited by several bacteria and microorganisms to survive harsh environments.

Given the above description, it should be rather clear that molding is rooted on an interaction mechanism that is *marker-based*, in that is uses a pheromone conceived for that purpose. It is based on *diffusion*in that the pheromone diffuses in the environment creating a gradient. It is *follow-through* since individuals are continuously driven by the gradient.

This kind of mechanism finds natural applications in robotics and team coordination — for example, a rescue team can decide to aggregate when particularly difficult conditions are met.

3.4 Embryogenesis

Embryogenesis is one of the major outstanding problems in the biological sciences. It concerns the fundamental question of how biological form is generated starting from an immense number of biological cells that self-organize their activities. One example of a mechanism common throughout development, in a wide range of species, is the use of morphogen gradients to determine positional information of cells. Cells at one end of the embryo emit a morphogen (protein) that diffuses along the length of the embryo. The concentration of this morphogen is used by other undifferentiated cells to determine their distance from the emitting end, and thus determining whether they lie in the head, thorax or abdominal regions [40]. Different morphogens are used for determining the position along other dimensions (e.g. dorsal-ventral axis). This mechanism has been widely studied — also experimentally — in the *Drosophilae* embryo.

In our taxonomy, this mechanism is *marker-based* in that it involves specialized chemicals (the morphogen gradients) to communicate. It is based on *diffusion* in that morphogen chemicals diffuse to enable long range communications. It is *trigger-based* in that the chemical induce one shot cell differentiation.

This mechanism can be fruitfully applied in a number of computer science scenarios such as in modular robotics and self-assembly. In these applications a swarm of robots uses virtual morphogens to achieve a final global shape.

3.5 Web Weaving

Web weaving is the metaphor in which individuals self-organize their activities to build collectively webs that are used both to easily roam across an environment ant to capture

preys passing nearby. The web weaving activity is based on the low-level task of connecting two ground spots with a dragline. The web, in the end, is composed by an aerial network of draglines, hooked to fixed spots on the ground. It is important to remark that, once fixed, a dragline provides a new path (the shortest one) between two spots. Similarly to the foraging metaphor, individuals roam randomly across the environment weaving a dragline between two random spots. However, at the same time, individuals are attracted by already existing draglines and prefer walking upon one of them rather than on the ground. This facilitates the chance of having a dragline beginning where another ends, and thus facilitates the web creation. In biological systems, the likelihood of walking upon a dragline has been carefully tuned. If it is too low, no web is built and all available space filled with disconnected segments. If it is too high, individuals are trapped in their own draglines and no real weaving occurs. This mechanisms is exploited by several spider species around the world. The *Anelosium eximius* which can be found in French Guiana is one of them. The individuals live together, share the same web and cooperate in various activities.

Following our taxonomy, at the core of this self-organized behavior, there is an interaction mechanism that is *sematectonic*, in that the very mechanism used to communicate is the web that individuals are building. It is *serendipitous*, in that individuals find each other silk draglines by chance. It is *follow-through* in that the web does not trigger one-shot reactions, but guides constantly individuals movements and weaving.

This mechanism can represent a solution to file retrieval in peer-to-peer systems and network routing. Virtual draglines can connect host or resources to allow them to be quickly discovered and retrieved later on.

3.6 Brood Sorting

Brood sorting is the metaphor in which individuals self-organize their activities to collectively sort items (such as eggs and microlarvea) found in the environment or in their nest. One self-organized behavior enabling this operation is that individuals just wander randomly and pick up and drop items according to the number of similar surrounding objects. For example, if an individual finds a large cluster of similar items together with a different one, it will most likely pick up the misplaced item and start roaming around. That individual will probably deposit its load in a region containing other items similar to the one he is carrying. The workers of the ant *Leptothorax unifasciatus* sort the colonys brood adopting this mechanism.

Again, in our taxonomy this is *sematectonic* in that it is the sorting process itself (i.e., first clusters) that drives coordination. It is *serendipitous* because individuals have to stumble onto a pile of (e.g. larvae) to coordinate. It is *trigger-based* in that a pile of items triggers one-shot pick-up or release of the items.

This mechanism can represent, for example, a solution to database organization and virus protection. A swarm of computational agents can review system resources, cluster them together and eventually detect outliers.

3.7 Flocking, Schooling and Herding

Flocking is a self-organized behavior that takes place when individuals are driven by three local forces: collision avoidance, speed matching and flock centering [61]. Schooling and herding are driven by similar forces [61]. Collision avoidance consists of individuals being programmed to pull away before crashing into one another. Speed matching relates to the attempt of the individuals to go at the same speed of their neighbors. Centering is the idea that individuals have a notion of the center of the flock and that they always try to move toward the center. More importantly, this coordinated movement happens without a leader — any individual can maneuver at any time [37]. More recently, Toner and Tu [69] have demonstrated that flocks never attain equilibrium, thus the flock is an entity that is continuing in characteristic but unpredictable in pattern formation. Flocking as well as schooling are common in birds and fish. As opposed to foraging, individuals here move less randomly. The attractiveness of flocks and schools to individuals is directly (but not linearly) proportional to its size [10]. In these systems, the presence of sentinels (which by no means are leaders) is also common. Sentinels in a herd are individuals that under certain environmental conditions are more sensitive to stimuli (eg. sound, smell, approach of danger) and react earlier than other individuals [48]. After the sentinel moves, others tend to follow due to the forces mentioned above.

Following our taxonomy, flocking is sematectonic in that is the very flocking formation (location of individuals) that drives coordination. It is based on diffusion in that information (visual information or wind turbulence) spread in space and can be sensed by other individuals at a distance. It is follow-through because it continuously drives motion.

This mechanism can represent a solution to motion coordination in a wide range of scenarios. For example a swarm of rovers or UAVs (Unmanned Airspace Vehicles) could use a flocking mechanism to roam across an environment.

3.8 Quorum

Quorum is the metaphor in which individuals self-organize their activities to sense how many other similar individuals are around them and react accordingly. The reaction normally relates to performing some action once a specific *quorum* is reached. This metaphor is possible because the actions of individuals diffuse some signal in their neighborhood and the density of the signal triggers the action. As in foraging, the signal is assumed to disappear over time, however quorum normally triggers an action that has a fixed duration in time (and that does not repeat too often), meaning that the speed at

which the signal fades is not as crucial as in foraging. Quorum is common in bacterial individuals such as the *vibrio fisheri*, a species of bioluminescent bacteria found in several niches in the marine environment; they reside in the light-emitting organs of certain species of squids and fish of the deep ocean. These individuals, use quorum to decide when to produce bioluminescence; they release a bioluminescent hormone while monitoring the levels bioluminescence in the environment.

Following our taxonomy, Quorum is rooted on an interaction mechanism that is sematectonic in that the action used to interact is also the one that individuals want to coordinate (e.g. light-emissions in *vibrio fisheri*). It is based on diffusion (e.g. in the *vibrio fisheri* the bioluminescent hormone diffuses across the population). It is trigger based; receive light information triggers actions (e.g. emitting light or change blinking rate).

This mechanism can represent a solution to coordination and synchronization in multi-agent systems. A similar approach has been used to synchronize the activities in a sensor network (see next section).

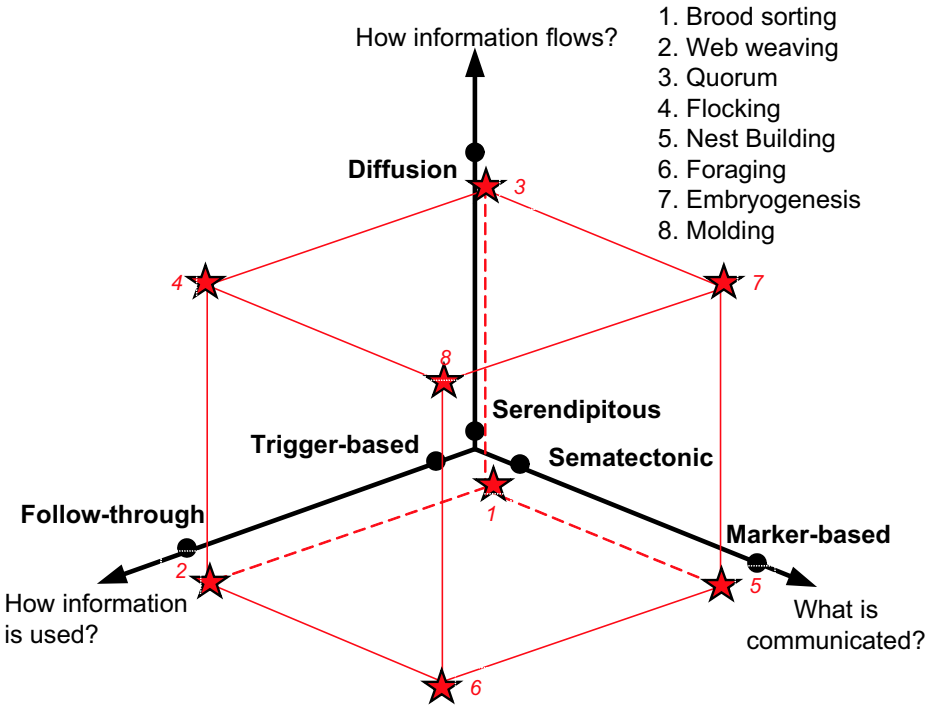

Fig. 1. Interaction mechanism and associated self-organizing metaphors.

4 Current Research

As anticipated in the previous section, several self-organization metaphors appear very suitable in handling a number of relevant computer science scenarios. If the robustness and flexibility of the biological systems employing such mechanisms could be transferred to software systems, then novel and much more reliable solutions to those scenarios could be realized. In the following we review applications for self-organization in several areas of computer science by referring to concrete existing work or by outlining possible applications that can be easily conceived.

Specifically, we identified 5 main areas that could naturally benefit from the presented self organized behaviors: *middleware*, *information systems and management*, *security*, *robotics*, and *network management*. Each area will be discussed in the next subsections.

4.1 Middleware

The development of middleware is essential to almost to all fields of applied computer science, but they are particularly important to distributed systems and multi-agent systems as they encapsulate mechanism to allow the coordination and organization of processes and data respectively. Self-organization in middleware is becoming more popular amongst the researchers as they are starting to see that the complexity of the problems in the field goes beyond the capabilities of standard approaches. In this section we divide the area of middleware into four application areas namely Grid Computing, Coordination Systems, Cache Replacement Systems, and Pervasive Computing.

Grid Computing

Description Although Foster [23] is seen as the father of Grid Computing with his many projects on the field, the general ideas can be traced as far back as the seventies with researchers at Xerox PARC (who envisioned a "worm" that could travel in a distributed system to utilize the computers idle time) More concrete proposals proposals, such as the Condor project [42] were proposed in the eighties. Grid Computing has been more of a "promise" in computer science than "reality". Despite the many success stories [22, 35, 47], the original idea of integrating heterogeneous systems in this nearly seamless framework has not really happened at the pace it was expected. Although the exact reason cannot be pin-pointed, it is fair to say that the complexity of such systems make them untreatable by professionals in the field. Grid frameworks propose to integrate databases, services, storage, and computational power under the umbrella of a grid to which users have access to in a similar way that they have access to power in power grid. If one need more storage space, one can have it if they pay for what it is worth. In order to realize such a framework, computing resources need to become a commodity in

which users do not need to worry about *how* it works but only about *what* they need and *how much* they can afford to pay for what they need. Research continues to develop grid frameworks because they offer a mechanism to solve grand challenge problems such as earthquake prediction, protein folding, etc. Current solutions are still very ad-hoc thus not being able to realize the full potential of the idea. Clearly there are other issues in Grid Computing such as security that are not less important but in one sense are orthogonal to the main challenges themselves and will have to be dealt with regardless of the solution given to the problems mentioned earlier.

Approach using Self-organization In order to see how self-organization can be used to improve the *status-quo* of grid computing, it is worth noticing that resource utilization is the *sine qua non* to performance in Grid environments: computers need to be well utilized, data need to be located where it is mostly needed, and services need to be easily found. At the same time this has to occur in a very dynamic environment. Recall that computers that are nodes in Grids can even be desktops with up-times that are not guaranteed — users can turn their machines on and off quite often. The resource utilization need to be adaptive to cope with such conditions while maintaining system with small overhead.

Self-organized metaphors such as foraging, molding, and brood sorting offers a way to re-think resource utilization in Grid frameworks. Foraging is a desirable metaphor to emerge solutions for finding and placing resources (e.g. services, data). A well implemented foraging approach can yield solutions that enable programs to locate resources with (near-)minimum effort. One can envision a grid framework in which histories of where services have been located in the past are used as a positive reinforcement to the decision of where to look for such services. Such histories can have a pheromone-like characteristics forcing the system to "forget" old histories. This idea has only been touched by researchers. For instance, Andrezejak *et Al.* [2] looked at the use of foraging as a distribution mechanism for services (see Figure 2).

Molding and brood sorting are more adequate to devise an approach to organize grid resources according to similarities. Brood sorting can be the basis for organizing data in certain locations in the grid according to similarities or based on a need by processes providing a emergent solution to the problem of load-balancing. Molding on the other hand can be used in conjunction to broad-sorting as a controlling mechanism to avoid over clustering. The differentiation mechanism in molding can serve the purpose of controlling the aggregation of data in only one location on the grid. As the cluster gets larger (thus moving towards becoming a bottleneck), molding can influence a new differentiation phase to form clusters in other locations in the grid. The emergent pattern is the formation of various clusters distributed on the grid (see Figure 2).

Coordination Systems

Description In the past 20 years tuple-based coordination models, and in particular Linda, have proven to be quite successful in tackling the intricacies of medium-to-large-scale open systems [26]. The atomic units of interaction in tuple-based coordination are tuples. A tuple is structured set of typed data items, i.e., a record. Coordination activities between application agents (there included synchronization) can take place via indirect exchange of tuples through a shared tuple space, i.e., a sort of shared dataspace that act simply as a tuple container. The coordination primitives provided to agents are a means to access and manipulate a shared tuple space. A tuple can be written in the tuple space or retrieved via an associative — pattern matching — mechanism. Agents can coordinate their actions — in a completely uncoupled way — on the basis of the presence or the lack of specific tuples. Currently, Linda's tuple-space model is incorporated in commercial middleware platforms such as Jini [3] and GigaSpaces [24]. However, it has not gained widespread acceptance as middleware for large-scale distributed systems. Currently, the dominating options for implementing distributed tuplespaces are described as follows:

- *Centralization* is a simple client-server distribution strategy where one specific server-machine operates the complete tuple space.
- *Partitioning* of tuple spaces is a strategy in which tuples with common characteristics are co-located in one of a a set of tuple-space servers.
- *Full replication* places complete copies of tuple spaces on several machines at different locations.
- *Intermediate replication* has been proposed in the early of Linda [16]. The schema bases on a grid of nodes formed by logical intersecting "busses". Each node is part of exactly one *outbus* and one *inbus*. Data stored is replicated on all nodes of the outbus, whereas searches are performed on the inbus. As one inbus intersects all outbusses, it provides a complete view of the tuple space.

None of the approaches described here scale well with the number of processes in the system. This severe engineering problem might have hindered the wide-acceptance of Linda in the past 20 years.

Approach using self-organization SwarmLinda [68, 49, 17, 51] is an attempt to transfer the *foraging* and *brood sorting* metaphors from natural agent systems to the implementation of large-scale distribution of a tuplespace.

The first area of SwarmLinda where abstraction from natural multi-agent systems is used is in the distribution of tuples amongst the nodes. Historically, tuples have been distributed using various *static* mechanisms as described above. In SwarmLinda the partitioning of the tuple space is dynamic and based on the concept of brood sorting used by ants. The individuals that operate here are tuple-ants. The environment is the network of SwarmLinda nodes. The state is the set of tuples stored thus far. A SwarmLinda implementation may use brood sorting in the process of tuple distribution to group tuples

based on their template which will lead to the formation of clusters of tuples. The power of this approach can be compared with the common partitioning scheme. Partitioning is based primarily on the use of a hash function to decide where the tuple should be placed. Hashing is not able to cope with failures and changes in the application behavior. Failures in certain nodes may be fatal to the system while changes in application behavior may require changes in the hash function being used.

The approach described above is able to improve the availability of the system without having to count on costly techniques such a replication of data. In the ant-based approach, there are no assumptions about the behavior of applications, there is no predefined distribution schema, and there are no special scenarios implemented to deal with failures in a node.

The distribution of tuples is only part of the problem — obviously these tuples need to be found. Ants look for food in the proximity of the anthill. Once found, the food is brought to the anthill and a trail is left so that other ants may know where food can be found. The ants know the way back to the anthill because they have a short memory of the last few steps they took and also because the anthill has a distinctive scent that can be tracked by the ants. In a tuple space context, foraging of ants can be abstracted by looking at tuples as food. The locations where the tuples are stored can be seen as the terrain while the templates are seen as ants that wander in the locations in search of tuples. The anthill is the process that executed the operation. The result of this is the emergence of application specific paths between tuple producers and consumers. Given that scents are volatile, the paths found can dynamically adapt to changes in the system — when consumers or producers join, leave or move within the system.

There are further occasions in the implementation of a Linda-like system to use self-organization. The named references give more details on how to show an adaptive behavior for collections of similar tuples by using tuple-ants and how to balance tuple- and template movement dynamically by using the ant-abstraction both for tuples and templates (see Figure 2).

Adaptive Web Cache Replacement Strategies

Description Part of the overall Web middleware for transferring information to users are Web caches. These sit between Web servers and clients and try to minimize network and server access to provide better response times to the end user. User-side caches act as proxies for servers and receive requests for information. They retrieve that information from servers and store a copy of it when transferring it to the client. When another client asks for the same information, it can be taken from the cache's store thereby minimizing the load for servers and the response time. Server-side caches are used to minimize load on servers for dynamic pages by storing the result for a retrieval for further accesses. They shield the server from the accesses that trigger processing, thereby minimizing the load of the servers and in turn lowering its response time for other dynamic

information items. The cache tries to optimize the *hit rate* which measures the ratio of information items found in the cache versus those those that had to be retrieved from the actual target server (sometimes also measures as the byte hit rate which weights hits with the size of hit objects). An important decision in that optimization has to be taken when the limited cache size is taken up by stored copies of information entities and some have to be selected for removal. The choice is critical in that it can degrade the caches performance if the "wrong" objects are removed, namely those that will be retrieved again soon. There are numerous techniques and strategies for cache placement and replacement [71]. All these have some benefits, however, they all exhibit their optimal performance only for specific characteristics of the stream of requests. Today, a Web cache implementation selects one of those strategies statically The well-known Squid cache for example, uses a LRU strategy for that. However, it would be interesting to have the cache dynamically select a strategy for the current stream of requests.

Approach using self-organization The foraging metaphor appears really suitable in this context. It is rather easy to imagine that cache hits can be used as food to feed virtual ants representing web cache entries (i.e., web resources). As long as a virtual ant get enough food (i.e., hits) it remains in cache. The ants that die out of food are replaced.To enforce adaptiveness, we can think of letting ants spread pheromones indicating where are the resources most likely to be accessed. Newly created ants could follow such pheromones trying to predict what resources will be accessed in the future, thus trying to crate novel relevant entries in the cache [21] (see Figure 2).

Pervasive Computing Systems

Description Pervasive computing introduces some peculiar challenging requirements in the development of distributed software systems: *(i)* since new agents can leave and arrive at any time, and can roam across different environments, applications have to be adaptive, and capable of dealing with such changes in a flexible and unsupervised way; *(ii)* the activities of the software systems are often contextual, i.e., strictly related to the environment in which the systems execute (e.g., a room or a street), whose characteristics are typically a priori unknown, thus requiring to dynamically enforce context-awareness; *(iii)* the adherence to the above requirements must not clash with the need for promoting a simple programming model possibly yielding light infrastructures. Unfortunately, current practice in distributed software development, as supported by currently available middleware infrastructures, is unlikely to effectively address the above requirement: *(i)* application agents are typically strictly coupled in their interactions (e.g., as in message-passing models and middleware), thus making it difficult to promote and support spontaneous interoperations; *(ii)* agents are provided with either no contextual information at all or with only low-expressive information (e.g., raw local data or simple events), which are difficult to be exploited for complex coordination

activities; *(iii)* due to the above, the results is usually an increase of both application and supporting environment complexity.

Approach using self-organization Field-based coordination relies on virtual computational fields (e.g., distributed data structures), mimicking gravitational and electromagnetic fields, as the basic mechanisms with which to coordinate activities in open and dynamic ensembles of application components. This enables components to spontaneously interact with each other via the mediation of fields and — as in physical systems — to self-organize their activity patterns in an adaptive way. All of this with the additional advantage that — unlike in real-world physical systems — one can shape fields according to any needed virtual physical law, to achieve a variety of coordination patterns in support of a variety of application goals. Accordingly to the description given in Section 2, we can say that field based coordination takes inspiration from both the *molding* and the *embryogenesis* metaphors. The main value of fields relies in that they can create distributed data structures providing agents with the right contextual information at the right location. Such a contextual information can profitably guides the agents' activities and supports their coordination. The model Co-Fields [46] and the middleware TOTA [45] support the spreading of fields (implemented by means of TOTA distributed tuples) across an ad-hoc network of pervasive computing devices. TOTA tuples — implementing fields — are characterized in terms of three parameters (C,P,M). C is the content of the tuple. It represents the information being carried on; such as the field properties and their magnitude. P is the propagation rule. It specifies how the tuple has to diffuse across the network. M is the maintenance rule, it specifies how the tuple should change upon network reconfigurations and external events. A clean API allows the creation of tuples — fields — and the injection of them in the network. Agents coordinate on the basis of the configuration of tuples — fields — sensed in their local environment. In a motion coordination application, for example, an agent could inject a tuple with an integer value that increases as the tuple propagates across the network. Another agent could follow the gradient of this tuple to discover and reach the location where the first agent is located (see Figure 2).

4.2 Information systems and management

Database Organization

Description For years, databases have been part of the mainstream computer science. From relational databases to temporal and object-oriented databases, the area had to evolve as applications demanded more from the database management systems (DBMSs). Despite all the research in the field, the main concepts are still based on IBM's original concept proposal — the first databases built on SQL in the market were Oracle version 2 and SQL/DS from IBMs and they could handle very small quantities of data.

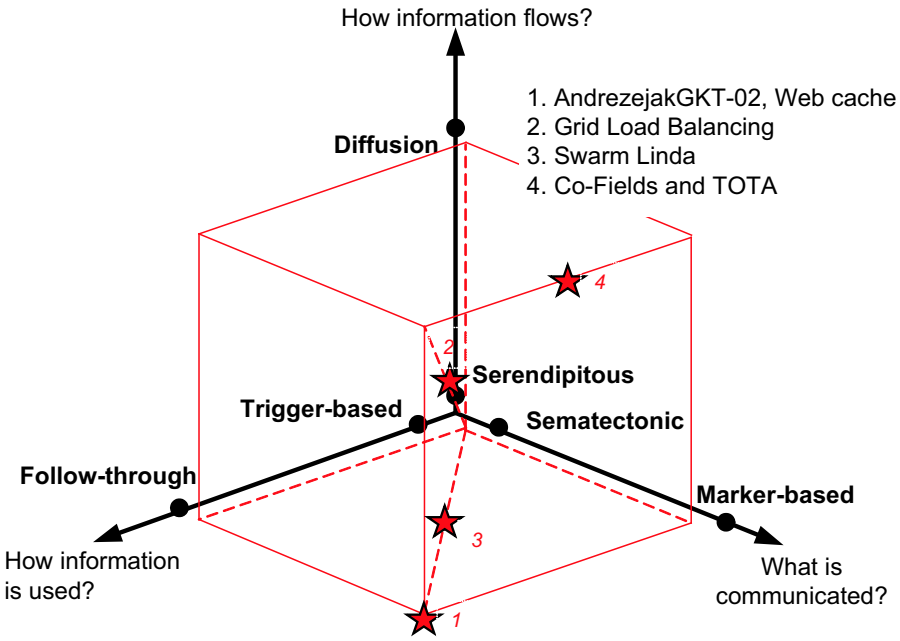

Fig. 2. Middleware infrastructures in the taxonomy space.

Nowadays, we are dealing with amounts of data that are hard to imagine and systems configurations that were never thought of by the first proponents of database ideas. In terms of amount of data, the CERN project on the exabyte (1 Exabyte $= 10^{18}$ bytes) database for Large Hadron Collider (LHC) experiments [27]. The job of physically organizing large amounts of data is quite a challenge to be resolved because no single machines is be able to store them — hence distribution is *sine qua non*.

A similar problem is the organization of small quantities of data that are scattered in small devices. In these scenarios, small devices (such as sensors) can hold small amounts of data that need to be organized in such a way so that queries can performed efficiently — new query processing systems may hold the key to efficiently used such devices.

Approach using self-organization Self-organized approaches are excellent for data organization. There are numerous metaphors in nature that demonstrate organization behavior that could inspire new databases. Distributed database systems (DDS) is a collection of logically interrelated databases distributed across various physical locations [53]. In addition to the database itself, we need a distributed database management system (DDBMS) which is the software part that allows the database to be handled. The data distribution aims at improving performance while maintaining transparency, reliability and cost savings. Self-organization inspired by *brood sorting* may be used to provide an adaptive distribution of data based on criteria based on the database tables, records(tuples), and even fields. If necessary, the criteria can also be physical (eg. secure data should be maintained in pre-defined secure sites)

The outcome of such distribution may be a distributed database system in which the distribution is adaptive to the activities being carried out, the size of the database, and other factors. One can envision an algorithm for data distribution based on brood sorting where each young correspond to an element of the database.[4] The elements of the database have incentives (positive feedback) to find locations where similar elements are located. To avoid bottlenecks we need to use scales of feedback where positive feedback becomes negative after some threshold. These concepts are very similar to the ideas of self-tuning databases such as the project COMFORT [72] which advocates that DBMSs should be DBA-free (Data Base Administrator). For instance, in large databases, issues such as load balancing can be self-regulatory so that the performance of the system is kept at good levels despite the its usage. Requirements such as these are well addressed by self-organized approaches as the ones described above in which elements of the database are able to move in a reactive way depending on the load of the system (see Figure 3).

Retail Stores

[4] The *element* depending on the form and granularity of distribution desired: vertical, horizontal, etc.

Description The problem of optimizing the process of manufacturing and retail (shopping) can save companies millions of dollars. The majority of the big shops use some mechanism to decide where the products are placed but few do this on-the-fly. The standard way of studying store efficiency, for instance, is to hire experts in the field and ask them to analyze the performance. In general these are very subjective and prone to the bias of the expert. Others prefer to do this study in-house to avoid the competition knowing of their secrets.

More recently, particularly in Europe, supermarkets started to experiment with new technologies to help them control stock, know more about store efficiency, and aiding on decision making. In the forefront of technology being used are the RFID (radio-frequency identification) tags. In the UK, Marks and Spencer and Tesco have pioneered the use of RFID for stock control. A more impressive attempt is the Metro Future Store, located in Rheinberg, Germany. With the help of RFID technology, the store is able to provide stock inventory, shelf filling information, implement a complex costumer information system, and payment system.

Approach using self-organization With all this technology becoming available it is surprising that not much has been published on how to address the issue of store organization based on costumer behavior. Self-organization may play an important role because people (costumers) in crowds behave very similar swarms of other animals.

Store organization can be made more efficient if we analyze the pastern of movement of costumers in the store. With RFID technology it is possible to build the path for every costumer and make decision on product placement in the store accordingly. Again here this would be similar to foraging where paths traversed by the costumer are being reinforced.

If a specific model is used, it may also be used to analyze the effect of changes before implementing them because the model can predict the outcome. Due to the self-organizing nature of people, what may be good for one store may not be necessarily good for others. A system based on foraging will always converge to the best to the current scenario, be it for different store, or different periods of the day — for instance it may be important see that certain aisles are only problematic during certain periods of the day.

The possibilities are almost infinite: one can do profiling of costumers — what they buy, when they buy; the store can make relation between products — if one buys product X it normally buys Y — which can be used to have special offers displayed on the path of the costumer for products that he is likely to buy even if not in his shopping list; the store may be able to save on energy cost with an intelligence light or cooling/heating system. Parts of the store that are not being used a lot may not require the same amount of cooling/heating as other parts (see Figure 3).

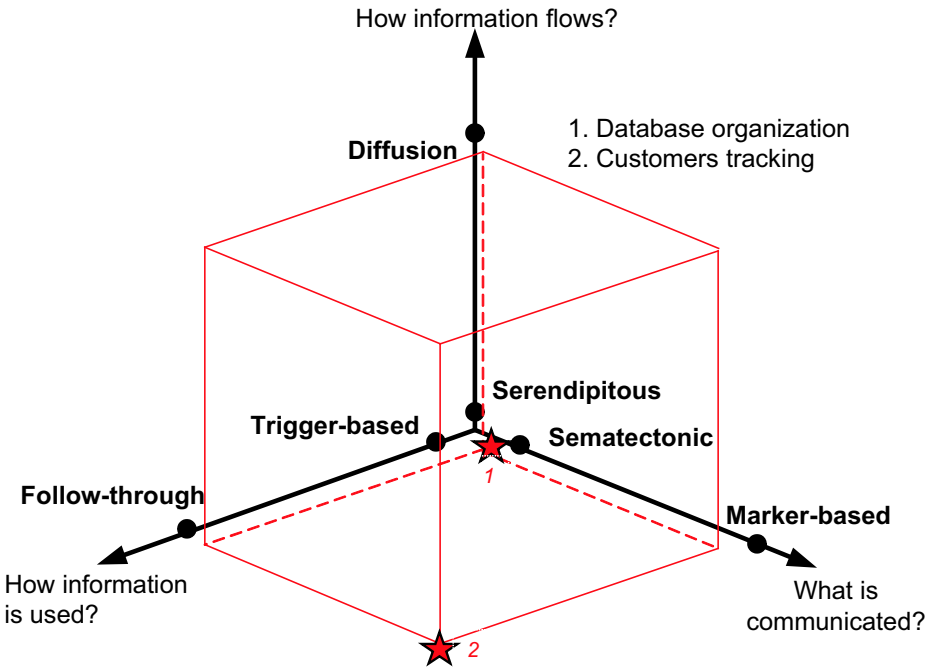

Fig. 3. Information systems in the taxonomy space.

4.3 Security

Trust Management

Description The distributed and connected world we live in requires us to share resources and experiences. One example of sharing experiences, that has become increasingly more important in our Internet world, is the distributed management of trust as introduced by Blaze *et Al.* [6] who frames the question of trust as: *Does the set C of credentials prove that the request R complies with the local security policy P?* At a less formal level, one can think of trust as a shared resources that identifies the degree of trustworthiness of a particular agent (physical or computational). Trust becomes a major concept in the design of large networked information systems. Even when information is transmitted securely and the sender is correctly authenticated, the process receiving the information is still left with the decision of what weight (level of truthness) to assign to the information passed. Can the sender be trusted to tell the truth? Does it[2] have a prior history of "good" information or has it been an unreliable source of information?

Currently, there is a broad area of research on building trust systems [5]. Many of these use some reputation system and many of them are centralized like the one used by `ebay.com`. However, the idea of trust is researched between humans, between artificial agents, and between humans and artificial agents (see T^3 Group for more details [29]).

Approach using self-organization Many works on trust management focus on issues of untrustworthiness vs. trustworthiness without scales. That is, the issue is discrete as one is either trusted or not. However, there are instances in which different levels of trust should be used as in a online banking application vs. an online bidding site. Clearly a trusted agent on an online bidding is not necessarily trusted to deal in a online banking application. In the agent world this is sometimes referred to also as *context* — trust depends on the context of the agent. So how can levels of trust be dynamically managed? One should note that the trustworthiness of an agent has a relation to the agent's age as well as the agent's history of previous activities. The memorization of what what has happened with levels of importance can be controlled dynamically using self-organized approaches based on a *quorum* metaphor. Take for instance the case of trust in online auctions sites where trust is based on the history of the users' activities on the site. Normally, these trust systems use symbols (such as stars and hearts) to represent the trustworthiness of the user according to his history. While somewhat effective, these systems are quite basic and consist of just counters of positive, negative, and neutral experiences of a particular agent with another. But a lot of other important information is left out: What about level of the experience? What about their age? That is, when did the experience take place? What about the trustworthiness of the person that contributing to the agent worthiness? — clearly an agent that cannot be trusted should not be

[2] The pronoun "it" here is used because the sender and receiver can also be an artificial agent

able to leave feedback that has the same weight as someone that is trusted. The difficulty of choosing such parameters makes the application ideal to self-organization. At the time of the writing of this survey, we could not find any published research on self-organization and trust. Still, a simple approach can be easily derived. Let us assume a $W_{p \to r}$ (read as the worthiness of person p from the point of view of person r) given by:

$$W_{p \to r} = (\varepsilon_{r \to p}) \times W_r$$

where $\varepsilon_{r \to p}$ represents the experience in the current transaction of r with person p. The equation above indicates that a person r can rate his last experience with a person p but this experience is weighted by its own current worthiness (W_r), where $0 \geq W_i, \varepsilon_i \geq 1, \forall i$. This new experience ($W_{p \to r}$) for a particular transaction is used in the equation below to calculate a new worthiness of p (W_p). The contribution of the new experience is controlled by a factor ρ:

$$W_p = (1 - \rho)W_p + (\rho)W_{p \to r}$$

The ρ represents the *forgetfulness* of the system — how fast the system should remember past experiences. The higher the value of ρ the longer past experiences are kept as an influence to W_p. The quorum metaphors comes from what W_p triggers. The system can be configured so that a specific value of W_p defines the probability that an agent is considered trusted. The specific values where the probability (threshold) is used may even be a function of all the W's in the system — if above a given value then the action than the probability is 1 that the agent is trusted. Conversely if below a given value the probability that an agent is trusted is 0. The above causes the emergence of relationship patterns among all agents starting from only point-to-point interactions between these agents (see Figure 5).

Malicious Code Protection

Description Humans cannot write perfect large software systems — that is fact. Even the most well developed critical system will contain some vulnerability. This fact, is what drives malicious code writers to develop programs that can effectively exploit these vulnerabilities. Our main hope against these techniques is the use of security tools such as anti-virus software. To better illustrate the main problems of the current state of the art, we will briefly make a comparison with biological viruses and immune systems. A virus is a non-cellular biological entity that can only reproduce when inside a host cell. The host cell then bursts and release copies of the virus that will then attach to new cells, enter the cells and continue the cycle of bursting, release of copies, and so on. Our immune system is able to cope with viruses by recognizing infected cells (called antigens) and destroying them. This process is very decentralized and very adaptive.

Our immune system is able to cope with unknown viruses using a process of refinement of current phagocytes (entities that try to attach and consume the antigens). The systems above only works because we do have an *adaptive* immune system — our body is able to generate protection against previously unknown malices. This is the opposite of what happens in a house fly for instance which is born with a pre-set number of defenses — a non-adaptive immune systems. Flies can evolve new protection mechanisms but only thought evolution of their generations. The scenario we find ourselves in computer science with regards to malicious code protection is similar to the one of a house fly. Our protection (anti-virus) are just like the immune systems of a fly while the malicious code (computer viruses) are like the malices. Because anti-viruses contain a pre-defined set of protection rules, malicious code writers can exploit this non-adaptive characteristic of the software and write effective malicious code. Again, like a fly, the protection system will only handle the new malice in new generations of the software — new anti-virus definition files.

Approach using self-organization Self-organization can be effectively used to provide adaptive protection against malicious code in the a similar way humans immune system can provide protection against antigens. The protection may works by distributing specialized agents in the network. These agents would recognize only few kinds of malicious code. The protection system (anti-virus for those who prefer the common name) enable agents to roam in search of known threats (to the system). Once an agent finds a malicious code it recognizes, it removes it from the system. This process is very close to *embryogenesis* because roaming agents may be seen as not-quite-specialized cells that only commit when a threat is identified. As in our immune system, these agents are attracted to the threat and will flow to areas of the network where the security has been breached. As a malicious code spreads, agents are able to replicate themselves to provide effective isolation of the compromised part of the system. Protection agents require the existence malicious code to replicate — low levels of a particular threat will cause the number of protection agents to be reduced (but never completely disappear). Agents could replicate if they sense large amounts of malicious code. This can be implemented by letting the malicious code act as a sort of pheromone that drives the replication of protection agents. As the pheromone concentration increases it may trigger the replication of the the protection agents. New agents can also be introduced in the system at any point to add new protections. New versions of old agents can also be introduced and programmed to remove the old agents (in a similar way we describe for policy updates later in this section). Most importantly, this systems needs to react quickly to new threats. Nature has shown us how this can be done. Each node of a network can contain a software stub which is in charge of letting the rest of the network know of unrecognized activities. This may be efficiently implemented by letting the stub diffuse the information on the network. Protection agents, being attracted by this diffused information, will flow to that location following the gradient of diffusion (similar to what is described for motion coordination and inspired in *molding* and *flocking*). The information collected by the software stubs may be used to aid industry in writing new protections. IBM has

explored ideas similar to what has been described above with their concept of artificial immune systems [38]. Their basic idea consist of having a mechanism to distinguish in a systems what is self (then them safe) from what is non-self (and consequently unsafe) (see Figure 5).

Distribution of Security Policies

Description A network security policy establishes a company-wide program of how users interact with a company's computer network, how the corporate computer architecture topology will be implemented, and where computer assets will be located — one aspect of network policies are firewall rules. Policies are one aspect of information which influences the behavior of objects within the system [64]. Human managers are adept at interpreting both formal and informal policy specifications and, if necessary, resolving conflicts when making decisions. However, the size and complexity of large distributed systems has resulted in a trend towards automating many aspects of management into distributed components. If policies are coded into these components they become inflexible and their behavior can only be altered by recoding [64]. Current approaches concentrate on ensuring transparency and compatibility. Transparency relates to ensuring that updates in the policy are as seamless as possible. Compatibility relates to ensuring that applications (benign) are not affected by the new policies. Little has been done on the effectiveness of the update of the policies, and how this update can be carried out in an autonomous fashion.

Approach using self-organization Menezes *et Al.*[50] have described a self-organized solution for policy distribution based on *foraging* with elements of *molding*. In order to understand how molding can be applicable, let us first assume a network in which all the nodes currently have a copy of the security policy Φ — one can assume that a node that does not have any policy is similar to one that has an outdated policy. Let us also assign one of the nodes as the initiator of the policy update process into a Φ_n where n is a value such that a version n is more current than all versions $< n$. The distribution algorithm based on molding works by making the security policies active and attracted by different (older) versions of policies. When a policy Φ_k is created, it looks at the neighbors and replicates itself to the neighbors that contain policies $\Phi_{<k}$. If the node finds nodes with policies $\Phi_{\geq k}$, the individual stops its activity since other individuals are already spreading on the network. Figure 4 shows a disconnection in the network and nodes (A, B and C) that were supposed to be updated but were not due to disconnection. They will be updated after reconnecting.

Molding is only utilized when there are failures on the network. Under normal circumstances, a foraging approach is used in the transition rule taking in consideration the policy version and the learned path from previous distribution. Our transition is given by:

Fig. 4. Initial node creates a new update for the security policy ζ_k that is then distributed on the network. Disconnection is handled automatically after the nodes reconnect.

$$p_{id}(t) = \frac{[\tau_{id}(t)]^{\alpha} \cdot [\delta_{id}(t)]^{\beta} \cdot [\eta_{id}(t)]^{\sigma}}{\sum_{h \in J_i} [\tau_{ih}(t)]^{\alpha} \cdot [\delta_{ih}(t)]^{\beta} \cdot [\eta_{ih}(t)]^{\sigma}}$$

In the equation above τ_{id} represents the learned behavior of the system. Each time a policy at node i chooses the neighbor $d \in J_i$ it reinforces the path $i \rightarrow d$ so as to indicate to other policies that the neighbor d was chosen at some point in the past. This controls the desirability of a node based on previous experiences of policies at that location. The term δ_{id} is responsible for the decision being made by the policy with regards to the version. The older the version (compared with the current one) at a given neighbor d, the more attractive d is. Last, the term η_{id} represents some static information about the network such as link reliability, or maximum available bandwidth.[3] This approach uses a local update technique each time a neighbor is chosen. This local update may modify the values of τ only. The value of δ is updated at defined intervals in which nodes "ping" their neighbors for that information. This requirement could actually be removed in favor of a more self-organizing approach which relies on the movement of the messages themselves to keep the values of δ updated for all neighbors of each individual node (see Figure 5).

4.4 Robotic Systems

The use of self-organization in robotics is current topic of intensive research. Several surveys exist covering this application of self-organization (eg. [15], and a few sections in [8]), here we try to complement what has been already described with a more up-to-date view of the field. In general, the power of self-organization with regard to robotics is about flexibility and robustness. A group of autonomous robots self-organizing their activities are able to adapt to dynamic situations, faults and glitches. For example, if a robot breaks down, the others can self-reorganize their activities and take over its duties. Bonabeau *et Al.*[8] emphasizes that there are several advantages in having self-organization implemented in robotic activities. In particular they discuss that randomness and fluctuations in individuals activities is an asset to problem solving, in that individuals become more flexible to changes, more reliable and more fault-tolerant. In the following, we will survey self-organizing approaches to tackle two exemplary application scenarios in multi-robot systems: motion-coordination and self-assembly.

[3] Network information is not currently being used but will be implemented as future work.

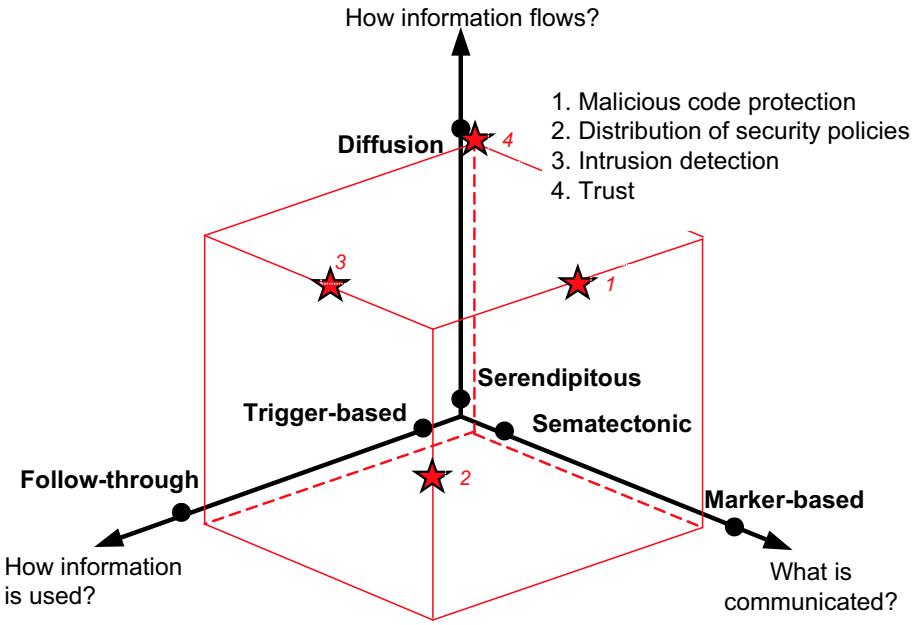

Fig. 5. Security mechanisms in the taxonomy space.

Motion Coordination

Description Motion coordination in robotics has been studied since long. The idea of this scenario is to have multiple robots — and more generally any kind of unmanned vehicles (aerial and ground, UAV and UGV respectively) — coordinating their respective movements to achieve some cooperative tasks. These include: exploring an environment, patrolling an area, moving according to specific tactics, surrounding a prey, etc. In the last few years, some very interesting proposals exploiting self-organization have been proposed.

Approach using self-organization A widely-exploited and interesting approach is based on the idea of having robots self-organize their movements on the basis of artificial potential fields that are digital analogues of the gravitational and electromagnetic fields [39]. Consider the case of a group of robots distributed in an environment, having to orient themselves and to coordinate their movements. To interact meaningfully, they have to exploit their sensing capabilities (e.g., cameras) to acquire contextual information. In addition, if they do not have direct means to communicate with each other's (e.g., they do not have wireless network interfaces) they necessarily have to coordinate with each other by detecting each other movements and adapt their behavior accordingly (in a sort of mediated, behavior-driven interaction). In these cases, robots could

take advantage from interpreting the environment and the other robots' behavior in terms of a field-based representation and acting on the basis of fields. As a simple example, an agent seeing an obstacle through its camera could represent it as a repelling field, whose strength increases as the perceived dimensions of the obstacle increase. As another example, it could perceive an object it has to carry as an attracting field. The main advantage of this approach is that it promotes a clean separation of concerns between the phase of processing sensorial stimuli (e.g., understanding that an object is an obstacle) and the phase of acting on the basis of such stimuli. Realizing motion coordination patterns with this approach is relatively easy. For example, if each robot emits an attracting field, the robots group at a single point. On the contrary, if each robot emits a repelling field, the robots tend to disperse in the environment — thus they explore the environment efficiently without overlapping. One of the most recent reissues of this idea, the Electric Field Approach (EFA) [32], has been exploited in the control of a team of Sony Aibo legged robots in the context of RoboCup competitions. In this approach, each Aibo robot builds a field-based representation of the environment from the images captured by its two head-mounted cameras (which also provides for stereo vision), and then decides its actions and movements by examining the gradients of the so-built environmental representation (see Figure 6).

Similar approaches to control robot movements have been proposed in [28, **?**]. In these proposals, a swarm of wirelessly interacting robots need to coordinate their movements to efficiently patrol an unknown environment. Here robots use their wireless network to actually spread distributed data structures representing the field abstraction. Each robot stores a piece of such a distributed data structure and moves on basis of the pieces contained in the robots in its neighborhood. These approaches are clearly inspired by the molding and flocking metaphors. Fields in fact are based on a marker-based interaction mechanisms. However, since their actual deployment depends on both the data structure and the current network topology (robots' positions), this approach has also sematectonic elements (see Figure 6).

In [55] a foraging inspired approach to coordinate UAVs is presented. This approach has been concretely implemented in a real-world scenario by spreading pheromones in a virtual data-space shared among the UAV agents. In this approach, a wirelessly accessible server holds a virtual copy of the environment the UAVs have to explore. UAVs connecting to that server can store pheromones at the location corresponding to their physical position. Moreover, they can access the resulting pheromone trails and move according to specific rules (see Figure 6).

Self Assembly

Description Robotic self-assembly refers to the application scenario of letting a large number of simple interacting micro-robots to arrange their respective positions and connect in a global structure that is suitable for a specific task. For example, in a pipe repairing task, a swarm of micro-robots could be able to navigate in a pipeline, recognize

the presence of holes, and self-assemble with each other so as to perfectly repair the pipe. In more futuristic scenarios, some kind of general purpose micro-robots could even self-assemble to create any kind of objects, from airplanes to tanks. From a software perspective, such a vision mainly requires the availability of complex coordination mechanisms and algorithms, so as to let the robots coherently assemble into the desired configuration.

Approach using self-organization In [34] a method is presented to organize the growth of a 2D structure in a swarm of mobile robots. Robots are autonomous, can only sense their local environment, and are largely interchangeable. Each robot is defined by a state value and a lookup table of transition rules. A transition rule specifies a condition under which a robot will connect itself to one of its neighbor robots. Connections, as prescribed by the lookup table, take place on the basis of a robot's internal state. Robots move randomly in a 2D square lattice around an initial robot, called seed, and when the conditions of a transition rule are met they attach themselves to neighbor robots. A similar approach is proposed in [18], where robots build a 3D structure using building blocks of different types and following building rules. Robots move randomly in a 3D lattice around an initial building block. They can sense the presence of building blocks or other robots within the local 3x3x3 lattice. When a robot finds a building block, it can pick it up and carry it toward the growing structure. Then, if the robot perceives a suitable environmental configuration (a match in its building rules), it deposits the block, making the structure grow. These two approaches clearly implement the Nest Building metaphor (see Figure 6).

In [7], each robot runs a simple finite state automaton in which state transitions are driven by the local state, the state of neighbor robots, their locations, and some external information. Communications are limited to the immediate neighborhood and a limited number of bits are exchanged at each time step. The goal is not to create an exact predefined shape, but a structure with the correct properties (structural, morphological, etc.). Any stable "emergent" structure that exhibits the desired properties is considered satisfactory, with no regard for the "optimality" or details of the resulting geometry. This approach is very similar to the nest building and web weaving metaphor (see Figure 6).

The approach presented in [65] enables robots to create regular spatial distributions like hexagonal and square lattices. Each robot is like a particle with a mass and it is subject to "artificial physical" forces enabling robots, detecting only nearby robots' distances, to spread in the space, creating regular lattices. This approach is clearly an implementation of the the molding metaphor. The approaches proposed in [44] and [66] bring this method further. In these approaches, the molding metaphor is realized by propagating multiple signals in the robots' ensemble. Robots react to the current signal configuration depending on their internal state. Following this approach, complex and articulated structures can be realized (see Figure 6).

Another interesting approach is presented in the Swarm-bots project [63, 25] which concentrate primarily on the use of self-organization in the assembling of structures

made of insect-sized active components. Individual components are called *s-bots*. A Swarm-bot is a group of them (of around 30) that self-assemble together. While the concentration of the project can be argued to be more on the hardware part of these system, the self-organization of the s-bots is also of prime importance. In particular, this project focuses on the novel concept of *functional* self-assembly. Robots have to assemble in such a way it is useful for the task they are trying to achieve. For example, they can form a long chain to overpass a crack in the terrain, or to climb a high step (see Figure 6).

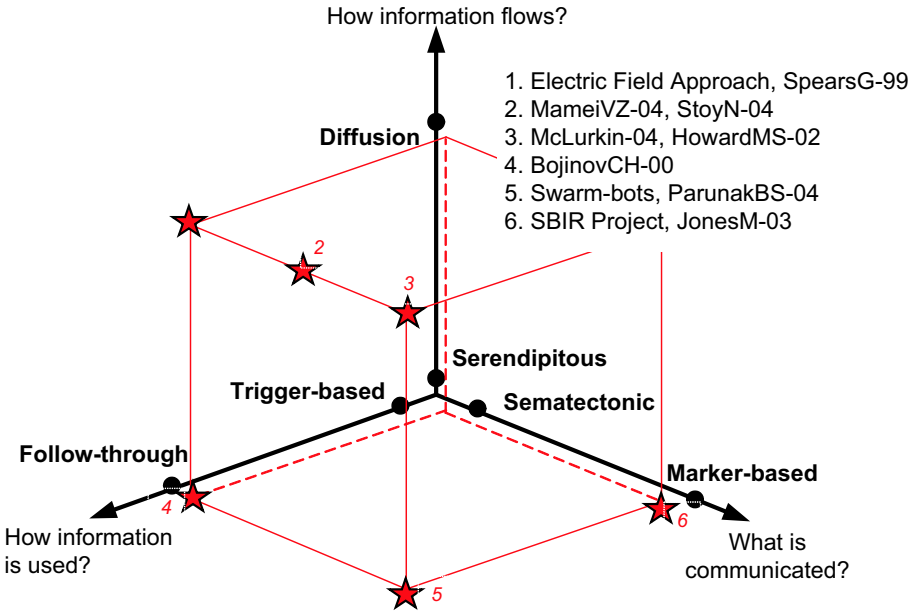

Fig. 6. Robot mechanisms in the taxonomy space.

4.5 Networks

Mobile Ad-hoc Networks

Description Wireless ad hoc networks and wireless sensor networks have recently received growing attention because of their promise of providing pervasive communication infrastructures that are cheap and trivial to deploy. However, they challenge current approaches to routing and network control, typically requiring manual configuration and suited for fixed or slowly changing network topologies. On the one hand, lacking

any centralized point of control, nodes in ad hoc networks must cooperatively self-organize routing and medium access functions. On the other hand, nodes in ad hoc networks may be mobile (consider, e.g., a network of PDAs carried by humans) or ephemeral (consider, e.g., the limited lifetime of battery-powered sensors), inducing continual changes in the network topology. For these reasons, new routing algorithms inspired by self-organization principles must be conceived.

Approach using self-organization In the last decade, a number of ad hoc network routing protocols have been proposed, which usually dynamically build a sort of routing overlay structure, and which then exploit this overlaid structure for routing. Among the others, (i) "Ant Routing" [9] and (ii) "Gradient Routing (GRAd)" [59] make self-organization metaphors in building the overlay particularly explicit.

Ant routing algorithms apply the foraging metaphor into the network routing problems. Derived by the forage model of real ant colonies, ant packets are used to explore the best route between specified source and destination nodes. The basic ideas of ant routing are described as follows. Instead of having a fixed next-hop destination, the routing table on nodes will have multiple next-hop choices for a destination, with each choice associated with a value indicating the goodness of choosing this hop as the next one to form the route. These values are initially equal and will be updated according to the ant packets pass by. Given a specified source node and destination node, the source node will send out some kind of ant packets based on the values on its own routing table: the more high the value of a link, the more likely the ant packet will be sent to that link. Those ants will explore the routes in the network recording the hops they have passed. When an ant packet reaches the destination node, the ant packet will return to the source node along the same route backward. Along the way back to the source node, the ant packet will change the routing table for every node increasing the value of the hop it comes from while decreasing the values of other links. Compared with the real ant foragers, changing routing table is just like laying down some virtual pheromone to affect the route of the subsequent ant packets. Since the route with higher value is favored, more ants will pick up that route, and the route will be strengthen. With this positive feed-back loop, we can expect a best path will be quickly found. Moreover, when a new best solution comes up, because of network dynamism, it could be identified and enforced by ant packets quickly. So ant routing is much more dynamic, robust and scalable (see Figure 7).

Gradient Routing is a routing algorithm specifically conceived for mobile ad hoc networks that takes inspiration form the molding metaphor. In very general terms, when a node "A" wants to send a message to a node "B," it actually floods the network with a field-like data structure that holds, the source id, i.e., "A," the message, and the number of hops from the source of the message to the current node. Such structure not only trivially hands off the message to "B" (since the message reaches all nodes), but also creates a sort of overlay field leading back to "A," to be exploited for further uses. If node "B" wants to reply, it can just send a message that follows the "A" field downhill

toward node "A." In this case no flooding is involved. The field-like distributed data structures created in this process can be used further also by other peers to communicate. However, such a field-based data structure has a limited life, in that changes in the network topology due to mobility may invalidate it. The next message sent by "A" will eventually help in rebuilding it (see Figure 7).

Sensor Networks

Description Sensor networks are a novel scenario comprising a huge number of microsensors wireless interacting to monitor an environment in a coordinated manner [20, 43, 56, 58]. The key issues being investigated in the area of sensor networks relate to the identification of effective algorithms and tools to perform distributed monitoring of activities by a cloud of distributed sensors in a physical environment. Representative goals pursued by these researches include tracing the position and movement of an object, determining the occurrence of specific environmental conditions, and reporting sensed data back to a base station in an efficient way. In general, these researches are indeed providing good insights on the theme of self-organized spontaneous interactions in amorphous ad hoc networks, and are leading to some very interesting results. Techniques for self-localization, self-synchronization of activities, and adaptive data distribution, all of which are of primary importance for any type of modern computing scenario, have been widely investigated.

Approach using self-organization Directed Diffusion [30] is a routing algorithm proposed in the sensor network domain. It makes an explicit use of metaphors like molding and ant foraging. Here the problem is how to collect sensed information from a vast network of sensors dispersed in an environment. The idea is that a workstation at the edge of the network can inject a field-like data structure (inspired to the molding idea) expressing an interest for a particular set of events ("query" field). Sensors are able to route relevant sensed information back to the workstation, exploiting the field data structure as a guide with a mechanism similar to the one described above for the Gradient Routing. Moreover, in this algorithm, the workstation can reinforce some part of the fields, applying a sort of reinforcement learning algorithm (similar to ants' pheromone reinforcement). This enables those sensors that have acquired "good" information to report back to the workstation, while allowing other sensors to forget about the "query" field (and thus save battery energy) if they are not able to provide relevant information (see Figure 7).

Another interesting application is presented in [41]. Here the goal is to guide a human user or a robot across an environment where a sensor network has been deployed. Sensors can detect dangers nearby and alert and guide users to stay away from dangerous areas. This "avoid danger" navigation has been implemented over a grid of Mica Motes sensors [57]. In particular, a sort of field-based approach — inspired to the molding metaphor — has been conveniently used. In such a potential fields approach, users

move under the actuation of artificial forces. The goal (i.e., the intended destination) generates an attractive potential field pulling the user to it. The recognized obstacles generate a repulsive potential which pushes the user away from them. The (negated) gradient of the total potential is the artificial force acting on the user. The direction of this force at a point in the landscape is the current best direction of motion from that point.

Amorphous Computing

Description The amorphous computing project [1, 13] focuses on a visionary scenario in which a massive numbers of identically programmed and locally interacting computational particles are randomly dispersed on a surface or mixed throughout a volume, and are possibly capable of sensing and affecting the environment. Particles are assumed to have limited resources and local information, and to be subject to faults. All particles are programmed identically, although each of them has an autonomous thread of execution, is capable of generating autonomously random numbers, and has an internal local state that can depend on past actions (i.e., a particle is a minimal agent). Particles can communicate with each other by short-distance radio connections, or through the substrate itself [13], or (in an even more visionary perspective) by emission of chemicals [73]. The key characteristic of amorphous computing, although it somewhat resembles cellular automata [75], is that particles have no a priori knowledge about the topology of the resulting communication network. No centralized source of information is available, and there are no global clocks, and no global beacons for triangulating positions. Thus, particles must engage in processes of self-organization for all their activities.

Approach using self-organization The research activity in amorphous computing is heavily inspired by biological systems, in particular by the embryogenesis metaphor. One of the main mechanisms in morphogenesis is that of morphogen gradient that, form a computer science perspective, resembles a distributed data structure that propagates through a network, by changing its values as it propagates. Accordingly, it can be reproduced in an amorphous network of computational particles and it can be used for a variety of purposes related to self-organization of activities. The key mechanism can work as follow. In a network of undifferentiated particles, an initial "source" particle, chosen by a cue from the environment or by generating a random value, creates a gradient by sending a message to its local neighborhood with the morphogen name and a value of zero. The neighboring particles forward the message to their neighbors with the value increased by one, and so on, until the morphogen has propagated through the entire population. Each particle stores and forwards only the minimum value it has received for a particular morphogen name; thus the morphogen value represents the shortest path from the source. The value provides an estimate of distance from the source: a point reached in n steps will be roughly distance nr away. The quality of this estimate

depends on the density of the particles and can be reasonably predicted for random distributions [52]. This very simple program can be used in powerful ways. By limiting the maximum value of a morphogen, one can create regions of controlled size. The morphogen can also be used to provide a sense of local orientation; a particle can compare values in its local neighborhood to determine the direction toward or away from the source. More than one particle could be the source for the same morphogen, in which case the morphogen value reflects the shortest distance to *any* of the sources. Thus, if a single particle emits a morphogen then the value increases as one moves radially away from the particle, but if a line of particles emits a morphogen then the value increases as one moves perpendicularly away from the line. In this way, complex spatial patterning can be created by positioning without any change to the particle program. The particles can also be programmed so that they can selectively choose which morphogens to propagate. Thus, particles in a particular state can act as barriers to specific morphogens, or as obstacles around which the morphogen must travel; or, similarly, morphogens can be limited to propagate only within certain spatial regions. As an additional example, the source particle can constantly produce a morphogen message, and have the morphogen value stored by any agent lose significance if not constantly reinforced. The result is that the morphogen values adapt as source particles appear and disappear (see Figure 7).

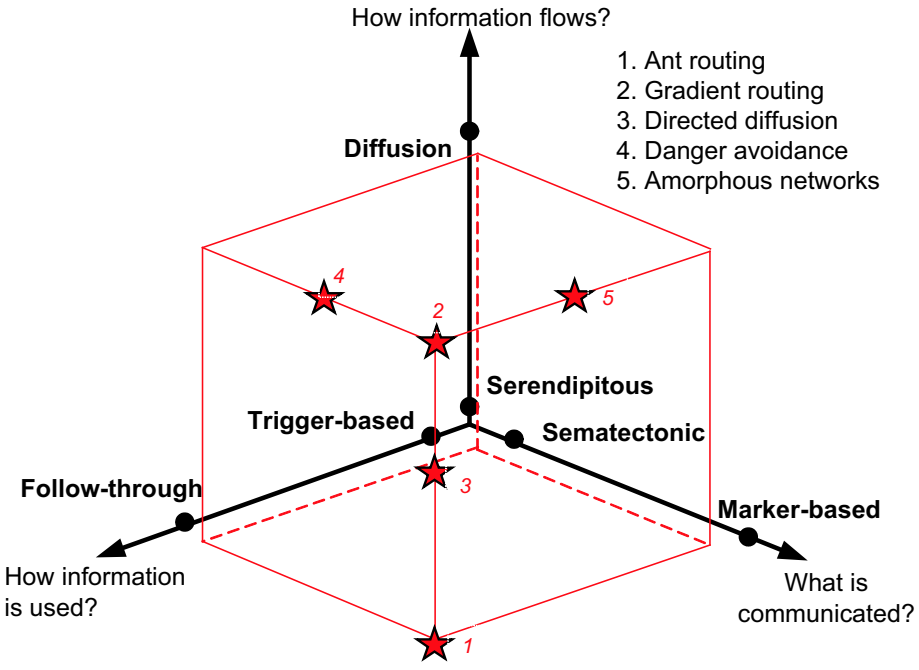

Fig. 7. Network approaches in the taxonomy space.

5 Conclusion

This paper introduced a taxonomy of self-organized metaphors and surveyed several applications that could benefit from the use of self-organization. Clearly, this paper had to leave some areas out due to space constraints. However, the purpose is to be diverse and survey applications that have different context.

What one should learn from this paper is that we have only scratched the surface of what self-organization and nature-inspired computing can do to computer science. As a matter of fact, self-organization appears to be the next "big concept" in science in general, with applications not only in computer science but in civil engineering, traffic control (for vehicles), medicine, to name but a few. In some of these fields we have already seen results such as buildings inspired by nature formations (what is being called *Bionic Buildings*) [12] and prototypes of intelligent control for vehicles [31] which could have been just easily taken from a flocking metaphor. In computer science the applications abound as this paper demonstrated. However, to allow further progress in this field, one may need to move from an ad-hoc approach to implementing self-organization to more structured and engineered approaches. We believe our taxonomy greatly contributes to this goal. Still, one may require the next step in the development process which is the study of methodologies and programming paradigms that reflect well the self organization of the metaphors described here.

Programmers and designers need also to change their state of mind and understand that in "self-organized design", they are more like conductors of an orchestra — they can set the pace, but in the end, the quality of the orchestra depends on the interactions between the musicians. In computer science, we are too accustomed to be in control and to have supreme power over our designs and implementations. In a self-organized approach, this is not possible given the complexity of the applications we are dealing with. Computer scientists must have a different state of mind and accepting uncertainty in their projects.

We believe this paper may be the start of this new way of thinking in computer science where we learn to design with uncertainty but we can be (probabilistically) confident that the system – at the global scale – will behave as required. We may be seen the birth of a new period in computer science — the self-organized period.

References

1. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman, and R. Weiss. Amorphous Computing. *Communications of the ACM*, 43(5):74 – 82, 2000.
2. A. Andrzejak, S. Graupner, V. Kotov, and H. Trinks. Algorithms for Self-Organization and Adaptive Service Placement in Dynamic Distributed Systems. Technical Report HPL-2002-259, Hewlett-Packard Laboratories Palo Alto, Sept. 2002.

3. K. Arnold, A. Wollrath, B. O'Sullivan, R. Scheifler, and J. Waldo. *The Jini specification*. Addison-Wesley, Reading, MA, USA, 1999.
4. A.-L. Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.
5. C. Bizer. Semantic Web Trust and Security Resource Guide. online, last seen 22.9.2004. http://www.wiwiss.fu-berlin.de/suhl/bizer/SWTSGuide/index.htm.
6. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173, Washington, DC, USA, 1996. IEEE Computer Society.
7. H. Bojinov, A. Casal, and T. Hogg. Emergent Structures in Modular Self-Reconfigurable Robots. In *Proceedings of the Intlernational Conference on Robotics and Automation*. IEEE CS Press, San Francisco, California, USA, 2000.
8. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Santa Fe Institute Studies in the Sciences of Complexity Series. Oxford Press, July 1999.
9. E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence. From Natural to Artificial Systems*. Oxford University Press, 1999.
10. C. Breder. Equations Descriptive of Fish Schools and other Animal Agregations. *Ecology*, 35:361–370, 1954.
11. M. Buchanan. *Nexus: Small Worlds and the Groundbreaking Science of Networks*. W. W. Norton & Company, 1 edition, May 2002.
12. R. Buswell, R. Soar, M. Pendlebury, A. Gibb, F. Edum-Fotwe, and A. Thorpe. 'Investigation of the potential for applying freeform processes to construction. In *Proceedings of the 3rd International Conference on Innovation in Architecture, Engineeing and Construction (AEC)*, pages 141–150, Rotterdam, The Netherlands, 2005.
13. W. Butera. Programming a Paintable Computer, 2002. PhD Thesis, MIT.
14. S. Camazine, J.-L. Deneubourg, N. Franks, J. Sneyd, G. Theraula, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton Univ Press, 2003.
15. Y. U. Cao, A. S. Fukunaga, and A. B. Kahng. Cooperative Mobile Robotics. *Autonomous Robots*, 4:7–27, 1997.
16. N. Carriero and D. Gelernter. The S/Net's Linda Kernel. *ACM Trans. Comput. Syst.*, 4(2):110–129, 1986.
17. A. Charles, R. Menezes, and R. Tolksdorf. On the Implementation of SwarmLinda. In *Proceedings of the ACM Southeastern Conference 04*, pages 296–297, 2004.
18. Cooperative Multi-Robot Control Architecture. http://www.dynamic-concepts.com.
19. M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.
20. D. Estrin, D. Culler, K. Pister, and G. Sukjatme. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, 1(1):59 – 69, 2002.
21. S. Floyd. Adaptive Web Cache. http://www.icir.org/floyd/web.html, 2005.
22. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.
23. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
24. G. T. Ltd. GigaSpaces platform. White Paper, February 2002.
25. G.C.Pettinaro, I. Kwee, L. M. Gambardella, F. Mondada, D. Floreano, S. Nolfi, J.-L. Deneubourg, and M. Dorigo. SWARM Robotics: A Different Approach to Service Robotics. In *Proceedings of the 33rd International Symposium on Robotics*, Stockholm, Sweden, 2002. International Federation of Robotics.

26. D. Gelernter and N.Carriero. Coordination Languages and Their Significance. *Communication of the ACM*, 35(2):96 – 107, 1992.
27. A. Hanushevsky and M. Nowark. Pursuit of a Scalable High Performance Multi-Petabyte Database. In *Proceedings of the 16th IEEE Symposium on Mass Storage Systems*, pages 169–175, San Diego, CA, USA, Mar. 1999. IEEE Press.
28. A. Howard, M. Mataric, and G. Sukhatme. An Incremental Self-Deployment Algorithm for Mobile Sensor Networks. *Autonomous Robots*, 13(2):113 – 126, 2002.
29. Institute of Cognitive Sciences and Technologies (ISTC). T^3 Group – Trust: Theory and Technology. http://www.istc.cnr.it/T3/index.html.
30. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking*. ACM Press, Boston, Massachusetts, USA, 2000.
31. P. A. Ioannou and C. C. Chien. Autonomous Intelligent Cruise Control. *IEEE Transactions on Vehicular Technology*, 42(4):657–672, Nov. 1993.
32. S. Johansson and A. Saffiotti. Using the Electric Field Approach in the RoboCup Domain. In *Proceedings of the Robocup International Symposium*, number 2377 in LNAI. Springer-Verlag, Seattle, Washington, USA, 2002.
33. S. Johnson. *Emergence: The Connected Lives of Ants, Brains, Cities, and Software*. Scribner, September 2002.
34. C. Jones and M. Mataric. From Local to Global Behavior in Intelligent Self-Assembly. In *Proceedings of the Conference on Robotics and Automation*. IEEE Press, Taipei, Taiwan, 2003.
35. N. H. Kapadia and J. A. B. Fortes. PUNCH: An architecture for Web-enabled wide-area network-computing. *Cluster Computing*, 2(2):153–164, 1999.
36. S. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford Press, 1993.
37. J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.
38. J. O. Kephart. A Biologically Inspired Immune System for Computers. In *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 130–139, Cambridge, MA, US, 1994. MIT Press.
39. O. Khatib. Real-time Obstacle Avoidance for Manipulators and Mobile Robots. *Journal of Robotics Research*, 5(1):90 – 98, 1986.
40. P. Lawrence. *The Making of a Fly: the Genetics of Animal Design*. Blackwell Science, 1992.
41. Q. Li, M. Rosa, and D. Rus. Distributed Algorithms for Guiding Navigation across a Sensor Network. In *Proceedings of the International Conference on Mobile Computing and Networking*. ACM Press, San Diego, Rhode Island, USA, 2003.
42. M. J. Litzkow, M. Livny, and M. W. Mutka. Condor: A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, San Jose, California, June 1988.
43. K. Lorincz, D. Malan, T. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, S. Moulton, and M. Welsh. Sensor Networks for Emergency Response: Challenges and Opportunities. *IEEE Pervasive Computing*, 3(4):16 – 23, 2004.
44. M. Mamei, M. Vasirani, and F. Zambonelli. Experiments of Morphogenesis in Swarms of Simple Mobile Robots. *Journal of Applied Artificial Intelligence*, 18(9 – 10):903 – 919, 2004.
45. M. Mamei and F. Zambonelli. Programming Pervasive and Mobile Computing Applications with the TOTA Middleware. In *Proceedings of the International Conference On Pervasive Computing (Percom)*. IEEE CS Press, Orlando, Florida, USA, 2004.

46. M. Mamei, F. Zambonelli, and L. Leonardi. Co-Fields: A Physically Inspired Approach to Distributed Motion Coordination. *IEEE Pervasive Computing*, 3(2):52 – 61, 2004.
47. V. Mann and M. Parashar. Engineering an Interoperable Computational Collaboratory on the Grid. *Concurrency and Computation: Practice and Experience*, 14(13–15):1569–1593, 2002. Special Issue on Grid Computing Environments.
48. G. Mead. *Mind, Self, and Society*. University of Chicago Press, 1934.
49. R. Menezes and R. Tolksdorf. A New Approach to Scalable Linda-systems Based on Swarms. In *Proceedings of ACM SAC 2003*, pages 375–379, 2003.
50. R. Menezes and R. Tolksdorf. A New Approach to Scalable Linda-systems Based on Swarms. In *Proceedings of the Symposium on Applied Computer*. ACM Press, Orlando, Florida, USA, 2003.
51. R. Menezes and R. Tolksdorf. Adaptiveness in Linda-based Coordination Models. In *Proceedings of the First International Workshop on Engineering Self-Organising Applications (ESOA 2003)*, number LNCS 2977. Springer, 2004.
52. R. Nagpal, H. Shrobe, and J. Bachrach. Organizing a Global Coordinate System from Local Information on an Ad Hoc Sensor Network. In *Proceedings of the International Workshop on Information Processing in Sensor Networks*, number 2634 in LNCS. Springer-Verlag, Palo Alto, California, USA, 2003.
53. M. T. Ozsu and P. Valduriez. *Readings in Distributed Computing Systems*, chapter Distributed Data Management: Unsolved Problems and New Issues, pages 512–544. IEEE Computer Society Press, 1994.
54. H. Parunak. Go to the Ant: Engineering Principles from Natural Multi-Agent Systems. *Annals of Operations Research*, 75:69–101, 1997.
55. V. Parunak, S. Brueckner, and J. Sauter. Digital Pheromones for Coordination of Unmanned Vehicles. In *Proceedings of the Workshop on Environments for Multi-agent Systems*. LNAI 3374, Springer Verlag, New York (NY), USA, 2004.
56. M. Paskin and C. Guestrin. A Robust Architecture for Distributed Inference in Sensor Network. In *Proceedings of the International Symposium on Information Processing in Sensor Networks*. ACM Press, Los Angeles, California, USA, 2005.
57. K. Pister. On the Limits and Applicability of MEMS Technology, 2000. Defense Science Study Group Report.
58. K. Pister. Invited Plenary Talk, 2003. International Conference on Distributed Computing Systems.
59. R. Poor. Embedded Networks: Pervasive, Low-Power, Wireless Connectivity, 2001. PhD Thesis, MIT.
60. M. Resnick. *Turtles, Termites, and Traffic Jams*. MIT Press, 1994.
61. C. Reynolds. Flocks, herds and schools: A distributed behavioral model. *Computer Graphics*, 21:25–34, 1987.
62. H. Rheingold. *Smart Mobs: The Next Social Revolution*. Perseus Books Group, Oct. 2002.
63. E. Sahin, T. Labella, V. Trianni, J.-L. Deneubourg, P. Rasse, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, and M. Dorigo. SWARM-BOTS: Pattern Formation in a Swarm of Self-Assembling Mobile Robots. In A. El Kamel, K. Mellouli, and P. Borne, editors, *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, Hammamet, Tunisia, Oct. 6-9, 2002. Piscataway, NJ: IEEE Press.
64. M. Sloman. Management Issues for Distributed Services. In *Proceedings of the IEEE Second International Workshop on Services in Distributed and Networked Environments*, pages 52–59, Whistler, Canada, June 1995. IEEE Press.

65. W. Spears and D. Gordon. Using artificial physics to control agents. In *International Conference on Information, Intelligence, and Systems*. IEEE CS Press, Rockville, Maryland, USA, 1999.
66. K. Stoy and R. Nagpal. Self-Reconfiguration Using Directed Growth. In *7th International Symposium on Distributed Autonomous Robotic Systems*. Springer-Verlag, Toulouse, France, 2004.
67. S. Strogatz. *Sync: The Emerging Science of Spontaneous Order*. hyperion Press, 1 edition, March 2003.
68. R. Tolksdorf and R. Menezes. Using Swarm Intelligence in Linda systems. In A. Omicini, P. Petta, and J. Pitt, editors, *Proceedings of the Fourth International Workshop Engineering Societies in the Agents World ESAW'03*, 2003. To appear.
69. J. Toner and Y. Tu. Flocks, herds and schools: A quantitative theory of flocking. *Physical Review E.*, 58:4828–4858, 1999.
70. F. Vertosick. *The Genius Within: Discovering the Intelligence of Every Living Thing*. Harcourt, 1 edition, June 2002.
71. J. Wang. A Survey of Web Caching Schemes for the Internet. *ACM Computer Communication Review*, 25(9):36–46, 1999.
72. G. Weikum, A. Mönkeberg, C. Hasse, and P. Zabback. Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering. In *Proceedings of the 28th Very Large Databases Conference (VLDB)*, pages 20–31, Aug. 2002.
73. R. Weiss, S. Basu, S. Hooshangi, A. Kalmbach, D. Karig, R. Mehreja, and I. Netravali. Genetic circuit building blocks for Cellular Computation, Communications, and Signal Processing. *Natural Computing*, 2(1):47 – 84, 2003.
74. Wikipedia. Self Organization. http://en.wikipedia.org/wiki/Self−organization.
75. S. Wolfram. *A New Kind Of Science*. Wolfram Media, 2002.